Release of the Alpha Test JNISpice Toolkit for Review

The Alpha Test JNISpice Toolkit is a prototype of "JNISpice": a product that provides an object-oriented Java interface to the C-language SPICE Toolkit CSPICE.

This prototype product is available to volunteer testers to enable review, with the aim of finalizing the specification and design of the product.

Reviewers of this prototype should already be very familiar with the SPICE system and be fluent in the Java language. Documentation provided in this product is not adequate to make the system easily understandable by SPICE or Java novices.

Below, we briefly discuss the review process. Following that are discussions of Alpha Test JNISpice Toolkit installation, the contents and status of the Alpha Test JNISpice Toolkit, regression testing, and running JNISpice-based user applications.

Regarding a Python interface to CSPICE

Those interested in PythonSpice, a Python and C-based analog of JNISpice that NAIF may develop, may wish to review the API of JNISpice, since NAIF's preliminary plan is to employ substantially parallel class designs for the high-level APIs of these products.

Reviewing the Alpha Test JNISpice Toolkit

Scope of review

All aspects of the JNISpice system specification, design, implementation, and documentation are ``fair game'' for review. Reviewers are asked to keep in mind that the purpose of the product is to to provide to Java applications convenient access to existing SPICE capabilities.

A list of known technical issues and deficiencies of the current system is provided below in Appendix A; this might serve as a starting point for discussion.

Since this system is a prototype under construction, much of the documentation that would be present in a standard SPICE Toolkit has not been provided. However, if expected functionality is missing and has not been noted in the ``known issues'' list, it would be helpful if reviewers were to point it out.

NAIF requests that reviewers run the JNITspice regression test suite on their host systems. See the section below titled ``Regression testing'' for details. NAIF would appreciate receiving logs of any failed tests.

Review time frame

NAIF would like to collect all review comments by May 1, 2010.

Contacting NAIF

Reviewers are requested to send comments to NAIF via e-mail to the JNISpice author Nat Bachman and NAIF manager Chuck Acton:

Nathaniel.Bachman@jpl.nasa.gov
Charles.H.Acton-Jr@jpl.nasa.gov

Installing the Alpha Test JNISpice Toolkit

Supported platforms

JNISpice requires that JDK version 1.5 or higher be installed on the host system.

The host system must be one for which a version of CSPICE is available. The one exception is the Mac Intel 64-bit CSPICE platform, which is not yet officially supported by NAIF, but for which a JNISpice Toolkit is available.

Each JNISpice Toolkit will work only on the host system for which it is targeted, so it's important to download the correct version. The supported platforms are

MacIntel_OSX_AppleC_Java1.5_32bit
MacIntel_OSX_AppleC_Java1.5_64bit
PC_Linux_GCC_Java1.5_32bit
PC_Linux_GCC_Java1.5_64bit
PC_Windows_VisualC_Java1.5_32bit
Sun_Solaris_GCC_Java1.5_32bit
Sun_Solaris_GCC_Java1.5_64bit
Sun_Solaris_SunC_Java1.5_32bit

Obtaining the Alpha Test JNISpice Toolkit

Reviewers may obtain Alpha Test JNISpice Toolkit packages from the NAIF server

naif.jpl.nasa.gov

via anonymous ftp. The packages are located in the platform-specific subdirectories of the path

pub/naif/misc/JNISpice

Each of the platform-specific subdirectories contains the subdirectories ``packages'' and ``executables.'' The ``executables'' directory contains the standard CSPICE utility programs. The ``packages'' subdirectory contains the Alpha Test JNISpice Toolkit packages.

Installation

Reviewers should select a path on their host systems under which to intall the JNISpice Toolkit. Below, we'll symbolize this path using the token

```
<mypath>
```

Reviewers should place the compressed tar file (Unix) or self-extracting archive (Windows)

```
JNISpice.tar.Z              [Unix]
JNISpice.exe                [Windows]
```

in

```
<mypath>
```

and then execute

```
importJNISpice.csh          [Unix]
JNISpice.exe                [Windows]
```

On Unix platforms, the installation script will extract files from the JNISpice package. On Windows platforms, the Toolkit will be extracted from the archive file. In each case, the C libraries and Java class files have already been built, so users need not perform any build tasks unless these files turn out to be incompatible with the host systems.

If necessary, reviewers can re-build the C libraries of the JNISpice Toolkit manually by executing the ``makeall.csh'' or ``makeall.bat'' script in the top-level path of the installed JNISpice Toolkit.

Note that the ``makeall'' script does not automatically recompile the JNISpice Toolkit's Java source code. NAIF does not expect that re-compiling the Java source code will be necessary, but if it is, reviewers can recompile that code by executing the ``mkjnijava.csh'' or ``mkjnijava.bat'' script in the path

```
<mypath>/JNISpice/src/JNISpice          [Unix]

<mypath>\JNISpice\src\JNISpice          [Windows]
```

JNISpice Contents

Installed JNISpice Toolkit

The installed JNISpice Toolkit is essentially a CSPICE Toolkit with JNISpice-specific modifications:

- -- The ``src'' path of the installed directory tree contains an extra subdirectory called JNISpice. JNISpice-specific C and Java source code files are located in and under this subdirectory. This JNISpice subdirectory and its subdirectories contain all of the javadoc-generated HTML pages constituting the JNISpice API reference guide.

- -- The ``src'' path contains a subdirectory for the C-language source code of the test utility library tutils_c.

- -- The file index.html in the path ``doc/html'' (Unix) or ``doc\html'' (Windows) of the installed directory tree contains a hyperlink to the ``Overview'' HTML page of the JNISpice API Reference Guide.

  This index file is the recommended starting point for viewing the JNISpice API documentation.

- -- This index file also has had two CSPICE-specific links deleted: those for the permuted index and ``most used'' functions.

- -- The ``lib'' path of the installed directory tree contains the JNISpice shared object library (Unix) or DLL (Windows), as well as the test utility library tutils_c.a.

- -- The ``exe'' path of the installed directory tree contains the regression test main program class file JNITspice.class.

JNISpice components

JNISpice provides Java applications access to CSPICE via Java's "native interface" (JNI) capability. JNISpice provides both a high-level, object-oriented Java interface, and a low-level Java interface consisting of native methods. Native methods are implemented via C-language wrappers for CSPICE functions.

JNISpice includes a regression test system called JNITspice. Reviewers are encouraged to run JNITspice to verify correct operation of JNISpice on their host systems. The code comprising JNITspice is independent of that providing standard SPICE functionality and does not affect the behavior of JNISpice-based applications.

All JNISpice Java and C source code files are included in this JNISpice Toolkit.

A complete N0063 CSPICE Toolkit is included as part of JNISpice. This Toolkit provides the CSPICE library required by the JNISpice Toolkit. It also provides the standard SPICE Toolkit utility programs.

As with all other SPICE Toolkits, JNISpice is a stand-alone product: it does not depend on or interact with any SPICE Toolkits you may have already installed on your system.

Documentation

The primary documentation for JNISpice is a set of HTML pages generated from JNISpice source code comments via the javadoc utility. This documentation can be accessed by following the hyperlink titled

Alpha Test JNISpice API Reference Guide

from the start page provided by the file

index.html

which resides in the path

<mypath>/JNISpice/doc/html [Unix]

<mypath>\JNISpice\doc\html [Windows]

of the installed JNISpice Toolkit. Here

<mypath>

denotes the host system path under which the JNISpice Toolkit is installed.

Following this link will cause the JNISpice Overview page to be displayed; all pages generated by javadoc can be accessed from this page.

Product status

This prototype system is at an "alpha test" level of maturity. At this stage, significant changes may still be made to the system's public interface, prior to the release of the system's first official version. As with all other SPICE Toolkits, once an official version has been released, the initial public interface will be supported for the life of the system.

A significant number of changes to the top-level Java API are planned: the known deficiencies listed below in the section titled "Technical Issues" will be corrected. The review process likely will result in further updates.

The design of the C wrapper layer is thought to be close to completion; only minor error handling enhancements are planned at this point.

At this early stage of JNISpice development, little effort has been expended on documentation of the API layer. Each class and method has at least a one-line description, but only a small subset of the key APIs have more detailed documentation. The section titled "Documentation" on the Overview HMTL page links to headers that contain example programs.

The included regression test system JNITspice is intended to exercise all of the methods of JNISpice's API-level classes: these are the classes in package spice.basic. However, JNITspice has not been reviewed by the NAIF team and cannot be guaranteed to completely test all of the code it's meant to cover.

Regression testing

The installed JNISpice Toolkit will contain the Java class file

```
JNITspice.class
```

in the path

```
<mypath>/JNISpice/exe                    [Unix]

<mypath>\JNISpice\exe                    [Windows]
```

where

```
<mypath>
```

denotes the path under which the JNISpice Toolkit is installed.

To run the regression test, cd to the path shown, then execute

```
[Unix 32-bit systems]

   java -cp .:../src/JNISpice -Djava.library.path=../lib JNITspice


[Unix 64-bit systems]

   java -d64 -cp .:../src/JNISpice -Djava.library.path=../lib
        JNITspice


[Windows 32-bit systems]

   java -cp .;..\src\JNISpice -Djava.library.path=..\lib JNITspice
```

When executed, the test suite will produce a log file having a name of the form

```
JNISpicenn.log
```

where nn denotes a version number. If all tests passed, a file named

```
passnnnn.log
```

will be generated. If any tests failed, a file named

```
ERRnnnn.log
```

will be generated.

Running JNISpice-based user applications

In order for JNISpice-based applications to run, the JNISpice shared object library (Unix) or DLL (Windows) must be loaded; in order for this to happen, the path of the library must be known to the ``java'' launcher.

Additionally, the location of the root of the directory tree containing the JNISpice classes must be known to the launcher.

Both of these paths can be specified as command-line arguments included in the ``java'' command. For examples, see the commands used to run JNITspice above in the ``Regression testing'' section.

The JNISpice classes that contain complete example programs in their in-line comments are identified in the JNISpice Overview. These programs may serve as a convenient starting point from which to construct custom JNISpice-based applications.

Users should note that run-time diagnostics produced by the Java system itself can be cryptic; in particular errors in the arguments of the ``java'' command may not result in what might be the expected diagnostics. So it can be helpful to inspect the command arguments carefully if one's application doesn't run.

Alpha Test JNISpice Technical Issues and Deficiencies

February 16, 2010

This document presents known questions and problems relating to the design, implementation, testing, and documentation of the Alpha Test JNISpice Toolkit.

For some issues, a brief rationale for a design or implementation choice is stated. The question of whether the choice is correct is implied.

Known API issues

String parameters

Many JNISpice classes declare public String constants. Better compile-time checking of user application code could be achieved if these constants were implemented via classes.

Note that many String constants cannot be conveniently implemented via enums because the constants' values don't conform to variable name syntax rules.

Lack of thread safety

All JNISpice native methods are declared static and synchronized. However, these methods must still be considered ``not thread safe'' since the underlying C code relies heavily on static variables.

It is unlikely that this limitation will be removed in a future version of JNISpice, since by its design, JNISpice relies on CSPICE for all but the most trivial computational capabilities.

Use of package-private or protected class members

Some classes such as Vector3 contain package-private members. The obvious alternate design calls for declaring these members private and providing accessor methods.

The rationale is that the selected privacy option simplifies the implementation, increases efficiency of simple operations, and does not pose a serious risk of misuse by application developers.

Package and class names

The JNISpice package names may be revised in the first official JNISpice Toolkit.

Many JNISpice class names are quite long and make source code formatting cumbersome. Some of these classes may need to be renamed as well.

Method names

NAIF plans to replace ``traditional'' matrix-vector arithmetic names such as ``mxm'' and ``mxv'' with mnemonic names such as ``add,'' ``sub'' and ``mult''; these names will be overloaded, and their meaning will be indicated by the object to which they are applied. For example, the method call m.mult(v) will invoke matrix-vector multiplication if m is of class Matrix33 and v is of class Vector3, while m0.mult(m1) invokes matrix multiplication if both m0 and m1 are of class Matrix33.

Are there problems with this approach, other than the lack of cross-language compatibility of names?

Output encapsulation classes

Certain classes that exist only to encapsulate lists of methods' outputs could be withdrawn. For example, the EllipsoidPointNearPoint class serves only to package the results of the point/ellipsoid ``near point'' computation: a surface point and an altitude. It's not clear that the small convenience of having the altitude computed automatically is worth the additional system complexity resulting from having an additional class.

Classes that encapsulate a combination of return values and ``found'' flags are not candidates for removal.

Variation of API patterns

Some closely related APIs have unexpected deviations from ``natural'' parallel functionality. For example, the EulerAngles class contains two ``getAngles'' methods: one that accepts an angular unit specifier and one that does not. The EulerState class contains only the version of the ``getAngles'' method that doesn't accept a unit specifier.

These variations must be rectified in the official JNISpice Toolkit.

GF search API design

Geometry Finder (GF) searches are set up via several steps, rather than implemented by one method call that requires as many as 25 input arguments.

While this interface pattern doesn't have a tight, parallel relationship to the corresponding CSPICE API pattern, it has the advantages of providing far simpler method signatures and greater modularity.

Lack of full support for intermediate-level GF APIs

Currently no APIs are provided to mimic the functionality of CSPICE's gfevnt_c and gffove_c APIs.

This is a deficiency that must be rectified in the official JNISpice Toolkit.

No default GF interrupt handling support

Unlike CSPICE, the JNISpice GF system doesn't trap control-C inputs by default when interrupt handling is enabled for the intermediate-level GF occultation search method.

Some type of default GF interrupt handling capability should be added in the official JNISpice Toolkit. Would a simple GUI interface for this be useful?

Default GF progress report uses console I/O

Should this be changed to a GUI interface?

No general dimension vector or matrix support

No APIs are provided to mimic the functionality of CSPICE's general-dimension vector and matrix functions.

Most of what are thought to be common applications of these functions are

supported by the classes Vector6 and Matrix66.

Should general-dimension APIs be added?

Visibility of support classes

Certain support classes, such as BodyCode and KernelVarDescriptor, perhaps should be made private or should be moved to a different package.

Exception classes

Should there be only one JNISpice exception class?

In any case, the use of JNISpice exception classes is currently not well organized and should be cleaned up.

Support for toString

Currently, a small set of JNISpice classes override class Object's toString method. Most, perhaps all JNISpice classes should override this method.

Formatting conventions used in JNISpice's toString methods need to be standardized.

Limited support for numeric forms of UTC

No top-level API support is provided for ``UTC seconds past J2000'' or ``UTC Julian date.''

Due to inherent defects of these UTC-based time representations, it doesn't make sense to represent them via JNISpice classes analogous to the existing JNISpice TDBTime or JED classes.

Currently, the only API that supports UTC seconds past J2000 is method deltet of the low-level class CSPICE.

Some sort of top-level API support is needed.

Use of constructors rather than factory methods

The heavy use of public constructors as the principal means of creating most JNISpice objects may appear to be a questionable implementation choice.

The main points in favor of this choice are:

- The constructor-based design is simple and appears to allow critical applications, such as state vector computations, to run at reasonable speeds.

- The public APIs are not allowed to change, so flexibility in

selecting new subclasses of returned objects in later versions of JNISpice appears to be of no value.

-- Factory methods can be added later if a need for application-specific memory management becomes apparent. This is considered to be unlikely.

Class SpiceWindow

Some aspects of the API of this class need work. Some common SpiceWindow operations, such as creation of windows representing time intervals, are awkward.

Class GFConstraint

Some aspects of the API of this class need work. Possibly this class should have subclasses for the different types of constraints.

Class SCLK

Possibly this class should be renamed---the current name leads to code that can be confusing to read.

Classes SPK and CK

Possibly the coverage summary methods of these classes should be made more parallel to their CSPICE counterparts.

No support for EK APIs

Due to expected sparse use of the EK APIs, support for these in JNISpice is considered a low priority.

Is this correct?

C-layer error handling

JNISpice C-layer utilities could do a better job of diagnosing invalid inputs, such as null pointers.

Documentation issues

Required Reading files

Currently only CSPICE versions of Required Reading files are available.

JNISpice versions are needed.

Permuted index

No analog of CSPICE's permuted index is available for JNISpice.

Is a JNISpice version needed?

``Most Used'' Routines

No analog of CSPICE's ``most used routines'' document is available for JNISpice.

Is a JNISpice version needed?

Class documentation

Most JNISpice classes have no documentation; full SPICE-quality documentation is needed.

Method documentation

The situation for methods is pretty much the same as for classes.

Package documentation

Packages lack overview documentation.

Host compatibility issues

The host system must use JDK version 1.5 or greater.

It is not known how many potential JNISpice users will be inconvenienced by this restriction.

Java header files

For the convenience of NAIF developers, the Java header files jni.h and jni_md.h are provided in the JNISpice Toolkit. This is not an acceptable design for the official JNISpice Toolkit: JNISpice build procedures should reference the header files provided by the host system's Java installation.

Test issues

Coverage

Coverage provided by the JNITspice regression test system has not been analyzed.

Memory leakage

Rigorous testing of JNISpice's dynamic memory usage has not been performed.

JNISpice utility code could do a better job of checking for memory leaks.

Lack of review

JNITspice test methods have not been reviewed by NAIF staff.

Implementation Issues

Lack of coding standards

NAIF has no official coding standard for Java source code.

One issue that would be addressed by such a standard is that of naming conventions for identifiers used in Java source code.

Class DAF

Implementation is awkward---may need re-writing.